Normalization Theory for XML

Normalization Theory for XML

Leonid Libkin[1]

School of Informatics, University of Edinburgh
libkin@inf.ed.ac.uk

Abstract. Specifications of XML documents typically consist of typing information (e.g., a DTD), and integrity constraints. Just like relational schema specifications, not all are good – some are prone to redundancies and update anomalies. In the relational world we have a well-developed theory of data design (also known as normalization). A few definitions of XML normal forms have been proposed, but the main question is *why* a particular design is good. In the XML world, we still lack universally accepted query languages such as relational algebra, or update languages that let us reason about storage redundancies, lossless decompositions, and update anomalies. A better approach, therefore, is to come up with notions of good design based on the intrinsic properties of the model itself. We present such an approach, based on Shannon's information theory, and show how it applies to relational normal forms as well as to XML design, for both native and relational storage.

1 Introduction

Data organization is one of the most fundamental topics in the study of databases. In fact, the concept of normalization was proposed by Codd [5] in 1971 – a mere year after he introduced the relational model. By 1974, the standard 2nd, 3rd, and Boyce-Codd [6] normal forms (2NF, 3NF, BCNF) had been developed. Bernstein's work on 3NF in the mid 1970s [3] is often viewed as the birth of database theory. It was understood very early by both database practitioners and theoreticians that having well-organized and well-designed databases is absolutely crucial for storing, querying, and updating data. Already in the 1980s, the standard normal forms such as 3NF and BCNF were covered by the majority of database texts.

After three decades of relational dominance, we have seen a new data format that is extremely widely used and can seriously challenge relational databases. Thanks to the proliferation of data on the web, much of it now appears in various markup language formats, of which XML is the most common one. Given the amount of data available in XML, it is natural to expect that some of XML designs will exhibit problems similar to those of relational designs, and indeed this is the case. As a simplest example, we can represent an arbitrary relational schema $R_1(A_1^1, \ldots, A_{n_1}^1), \ldots, R_1(A_1^k, \ldots, A_{n_k}^k)$ in XML by means of the following DTD D_1:

$$db \to R_1, \ldots, R_k$$
$$R_i \to tuple_i^*, \quad i \le k$$

that declares $A^i_1, \ldots, A^i_{n_i}$ to be the attributes of $tuple_i$. This way, a bad relational design translates into a bad XML design, inheriting its problems such as redundancies and update anomalies. But there are other ways to have designs prone to update anomalies due to the *hierarchical* nature of XML. Consider, for example, the following DTD D_2 for storing information about conference publications:

$$db \rightarrow conf^*$$
$$conf \rightarrow paper^*$$
$$paper \rightarrow author^+, title, year$$

with *author*, *title*, and *year* elements each carrying an attribute with its value. Now suppose we know – and this is a reasonable assumption – that all papers in a conference have the same publication year. Then the *year* information is redundant, as it is stored repeatedly for all papers in the conference. In addition it is likely to lead to update problems: if a year needs to be changed, one cannot do it just once; instead it needs to be changed for every paper in the conference.

To build foundations of good XML design, we need to answer the following two questions:

1. How do we recognize poor XML designs, and how do we convert them to good designs? In other words, we want to develop a theory of normalization for XML.
2. What constitutes a good XML design? In other words, we want to formulate criteria for good XML designs. In the relation case, one usually appeals to the intuitive notions of redundancy and update anomaly. But this approach is problematic in the case of XML for three reasons:
   - First, due to the complicated hierarchical structure of XML documents, it is harder to see when a schema contains redundancies.
   - Second, the notion of an update is not nearly as clean as the notion of relational updates, which makes it hard to say what constitutes an update anomaly (especially in the absence of a universally accepted notion of XML updates).
   - Third, there is no query language with the same yardstick status as relational algebra has for relational databases. The process of normalization needs to be lossless (meaning that the original data can be recovered from a differently designed schema). Thus, the notion of losslessness depends on a query language.

   Thus, we need an approach based on "standards-free" XML concepts, that is, concepts that will not change even if the W3C comes up with a new query or update language for XML tomorrow.

Several XML normal forms have been proposed recently, see, e.g., [1, 18, 17, 19, 8]. They differ in terms of schema and constraint description, but are based on essentially the same set of transformations, first proposed in [1]. As for the work on justification of XML normal forms, an approach was proposed in [2] based on information theory. The idea of the approach is that we measure the amount of redundancy in a design *regardless* of any query/update language for

a data model. This approach, when applied to relational design, confirms our intuitive view of which designs are good [2, 12], and then it can be applied in the case of XML to reason about XML designs for both native [2] and relational storage [13].

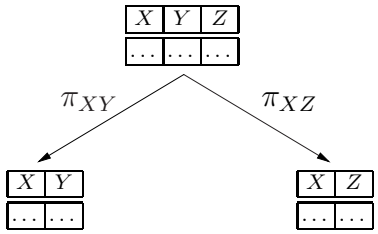
We give a brief survey of these developments. We start with a quick overview of relational normalization. After that, we introduce the main idea of XML normalization. We then present the intuition behind the information-theoretic approach to normalization, and show how it justifies commonly used relational normal forms. Finally, we analyze the implications of the information-theoretic approach to XML design.

2 Relational normalization: a brief reminder

We give an overview of two normal forms based on functional dependencies (FDs): the third normal form 3NF, and the Boyce-Codd normal form BCNF.

If integrity constraints are specified as FDs, the main cause for problems in a design is a functional dependency $X \to Y$ in which the left-hand side X is not a key. For example, if we have three attributes A, B, C and an FD $A \to B$, then a given value a of A can be associated with an arbitrary number of values of the C-attribute, i.e. we can have tuples $(a, b, c_1), \ldots, (a, b, c_n)$ in a relation, but the value of the B-attribute must be the same in all of them, as it is determined by a. Hence, we store b unnecessarily many times. Besides, an attempt to update b leads to problems as it needs to be updated in all of the tuples – otherwise the database would be inconsistent.

A standard solution in this case is to split a schema into two; in this case, into AB with a key dependency $A \to B$, and AC. In general, if we have an FD $X \to Y$, and Z is the set of attributes that are not dependent on X, one splits the schema XYZ as follows:



Thus, a "bad" FD $X \to Y$ is translated into a key dependency $X \to Y$ in the relation with XY attributes, and generates a foreign key from the relation with XZ attributes.

If we have a relational schema given by a set of attributes U and a set of functional dependencies F, we write F^+ for the set of all FDs logically implied by F. We say that a schema is in *BCNF* if, for every nontrivial FD $X \to Y$ in F (i.e. $Y \not\subseteq X$), it is the case that X is a key; in other words, $X \to U \in F^+$.

Intuitively, BCNF completely eliminates all the redundancies, as it eliminates "bad" FDs, and replaces them by keys and foreign keys. It does suffer from one

problem, however. Consider a schema with three attributes A, B, C and two FDs $AB \to C$ and $C \to A$. This schema is not in BCNF, since C is not a key. If we split it into AC and BC, we lose the FD $AB \to C$. In other words, the decomposition is not *dependency-preserving*, and in fact no lossless decomposition of this schema is. Hence, it is impossible to achieve complete elimination of redundancies and dependency-preservation at the same time.

If dependency-preservation is important (and it usually is, since it is important to maintain consistency of the data), one may have to settle for less than BCNF. Recall that an attribute is called *prime* if it belongs to a candidate (minimal) key. Now assume that in a schema we allow two conditions for a nontrivial FD $X \to Y$: either X is a key (as in the case of BCNF), or every attribute in $Y - X$ is prime. This is the definition of the third normal form 3NF (actually, this is not the original definition but a reformulation by [20], which is most commonly used these days).

A good property of 3NF is that every relational schema admits a lossless dependency-preserving 3NF decomposition. However, such a decomposition is not guaranteed to eliminate all the redundancies, but it restricts them to values of prime attributes. 3NF is a very common database design used in practice [15].

3 Measuring the amount of redundancy

Our goal is to provide a way of reasoning about XML designs without appealing to the notions of queries and updates for XML documents. Before introducing such techniques for XML, we would like to test them in the well-understood relational case, where we know what constitutes a good design.

The main idea of our approach, which was first proposed in [2], is as follows. Given an instance R of a schema with attributes $A_1, \ldots, A_n$ and FDs F, we define a notion of *relative information content* of a position p in R with respect to F. It will be denoted by $\textsc{Ric}_R(p|F)$. Here a position is identified by a tuple and an attribute. We define it in such a way that

$$0 \;\leq\; \textsc{Ric}_R(p|F) \;\leq\; 1,$$

with $\textsc{Ric}_R(p|F) = 1$ saying that position p carries no redundancy whatsoever. In general, the less the value of $\textsc{Ric}_R(p|F)$ is, the more redundancy this position p carries.

This notion is defined using the concept of entropy, more precisely, a certain conditional entropy. The notion of entropy was used in the past to reason about database constraints [7, 14], but it is a bit of challenge to make it relative to a set of constraints F. The general idea is as follows. We want to measure the amount of information in p with respect to an arbitrary set P of positions in the instance R. This way we account for all possible interactions between p and sets of positions in R, and then we take the average such amount of information as the value of $\textsc{Ric}_R(p|F)$.

To measure the amount of information in p with respect to a set of positions P, assume that we lose the value in position p, and that we have a set of k possible

values $v_1, \ldots, v_k$ to choose this value from. We shall assign a certain probability $\pi_i(P)$ of picking the right value to each of the v_i's – this is the probability that v_i is a possible value for position p, given the information provided by positions P. We then look at the entropy of this distribution:

$$\textsc{Ric}_R^k(p, P|F) \;=\; \sum_{i=1}^{k} \pi_i(P) \log \frac{1}{\pi_i(P)}.$$

Note that this value is dependent on k, the number of possible values to put in position p.

The entropy tells us how much information is provided by a certain random event. For example, if there is only one way to replace the missing value by some v_i, then $\textsc{Ric}_R^k(p, P|F) = 0$, meaning that the information content of position p is 0, and that the value is redundant as it can be inferred from the rest. The opposite case when all the values v_i's are possible with equal probabilities $\pi_i(P) = \frac{1}{k}$. This is the the least redundant case, when we can infer nothing about the value in position p. In this case, $\textsc{Ric}_R^k(p, P|F) = \log k$, the maximum value of an entropy of a discrete distribution on k elements.

Now our measure (almost) is the average value of $\textsc{Ric}_R^k(p, P|F)$ over all sets P of positions:

$$\textsc{Ric}_R^k(p|F) \;=\; \frac{1}{2^{N-1}} \sum_{P \subseteq Positions(R)} \textsc{Ric}_R^k(p, P|F),$$

where N is the total number of positions in R (i.e. the number of tuples in R multiplied by the number of attributes in R).

To compute the $\pi_i(P)$'s, we need the FDs in F. Now suppose that the value if position p is v_i, and we lose the values in positions P. We now look at the ratio of the number of value assignments to position in P (from the same set $v_1, \ldots, v_k$) that make the resulting instance satisfy all the FDs in F. These are, essentially, the $\pi_i(P)$'s (in addition one needs to normalize these values to ensure that $\sum_i \pi_i(P) = 1$).

We have almost defined our measure; the only problem is that $\textsc{Ric}_R^k(p|F)$ depends on k. Since the domain of values is assumed to be countably infinite, as the measure $\textsc{Ric}_R(p|F)$ we take the limit of the ratio of $\textsc{Ric}_R^k(p|F)$ and the maximum entropy for a discrete distribution on k values, i.e.

$$\textsc{Ric}_R(p|F) \;=\; \lim_{k \to \infty} \frac{\textsc{Ric}_R^k(p|F)}{\log k}.$$

It was proved in [2] that this limit always exists (in fact it exists for far more general classes of constraints than FDs), and thus can be taken to be the relative information content of a position in an instance with respect to a set of constraints F.

Some basic facts about the measure $\textsc{Ric}_R(p|F)$ from [2]:

– The value of $\text{RIC}_R(p|F)$ is independent of the syntactic representation of the FDs in F. That is, if F and G are two sets of FDs and $F^+ = G^+$, then $\text{RIC}_R(p|F) = \text{RIC}_R(p|G)$.
– $0 \leq \text{RIC}_R(p|F) \leq 1$.
– If $F = \emptyset$, then $\text{RIC}_R(p|F) = 1$ (if there are no constraints, there is nothing to tell us that the schema may not be well-designed).

4 Applying the measure: relational designs

We now use the information-theoretic measure to define well-designed schemas.

Definition 1. *A relational schema given by a set of FDs F is* well-designed *iff* $\text{RIC}_R(p|F) = 1$ *for every instance R of the schema and every position p in R.*

In other words, the schema is well-designed if no position in any instance of the schema admits any redundancy.

Theorem 1 (see [2]). *A schema given by FDs is well-designed iff it is in BCNF.*

This confirms our intuition that BCNF completely eliminates redundancies. This result was further extended in [2] to deal with other classes of constraints such as multi-valued and join dependencies, and justify normal forms such as 4NF [9].

But what about the normal form most commonly used in practice, i.e. 3NF? The first result looks rather discouraging: 3NF schemas may admit an arbitrarily high amount of redundancy (demonstrated by arbitrarily low values of the measure).

Proposition 1 (see [11]). *For every $0 < \varepsilon < 1$, one can find a 3NF relational schema given by a set of FDs F, an instance R of that schema and a position p in R such that* $\text{RIC}_R(p|F) < \varepsilon$.

However, the situation is not as bad as it might seem. First, the result requires schemas with a large number of attributes. More importantly, it has been known for a long time [20] that not all 3NF designs are equally good: for some schemas already in 3NF, better 3NF designs can be obtained by applying the standard 3NF synthesis algorithm [3].

3NF designs guarantee the integrity of the database. One may ask whether 3NF is the best choice of a dependency-preserving normal form. That is, if we look at all normal forms that guarantee dependency-preservation (hence excluding BCNF), is it 3NF that has the least amount of redundancy?

The answer to this is positive. Assume that $\mathcal{NF}$ is some dependency-preserving normal form: i.e., every schema admits a lossless dependency-preserving decomposition into $\mathcal{NF}$. We define the *guaranteed information content* provided by $\mathcal{NF}$ as the largest number $c \in [0, 1]$ such that every schema may be decomposed

into $\mathcal{NF}$ in such a way that in all instances R of the decomposed schema and all positions p, the information content $\textsc{Ric}_R(p|F)$ is at least c.

If $\textsc{Ric}_R(p|F) \geq c$, then $1-c$ is the *price of dependency preservation*, denoted by $\textsc{Price}(\mathcal{NF})$: that is, the minimum amount of information content one *must* lose due to dependency preservation. The following theorem shows that among normal forms that guarantee dependency preservation, 3NF is the one with the least amount of redundancy.

Theorem 2 (see [12]). $\textsc{Price}(\text{3NF}) = 1/2$. *Furthermore, if $\mathcal{NF}$ is an arbitrary dependency-preserving normal form, then* $\textsc{Price}(\mathcal{NF}) \geq 1/2$.

Furthermore, we can analyze "good" 3NF schemas produced by the standard synthesis algorithm. We refer to them as 3NF^+ schemas (they can be syntactically characterized [20], but for our purposes it suffices to think of them as 3NF schemas that cannot be further decomposed using the standard synthesis algorithm of [3]).

To compare 3NF and 3NF^+ designs, we use a new concept of a *gain of normalization* function. To define it, assume that we have a condition $\mathcal{C}$ on schemas consisting of FDs. We first define the set of possible values of $\textsc{Ric}_R(p|F)$ for m-attribute instances R of schemas satisfying $\mathcal{C}$:

$$\mathcal{POSS}_{\mathcal{C}}(m) = \{\textsc{Ric}_R(p|F) \mid R \text{ satisfies } F, \quad F \text{ satisfies } \mathcal{C}, \\ R \text{ has } m \text{ attributes}\}.$$

We are now interested in the lowest possible value in such a set, i.e. $\inf \mathcal{POSS}_{\mathcal{C}}(m)$ (typically these sets are dense subsets of intervals $(\varepsilon, 1]$, so the infimum – ε – is well-defined). For two normal forms $\mathcal{NF}_1$ and $\mathcal{NF}_2$, the gain of normalization function $\textsc{Gain}_{\mathcal{NF}_1/\mathcal{NF}_2} : \mathbb{N} \to \mathbb{R}$ is

$$\textsc{Gain}_{\mathcal{NF}_1/\mathcal{NF}_2}(m) \;=\; \frac{\inf \mathcal{POSS}_{\mathcal{NF}_1}(m)}{\inf \mathcal{POSS}_{\mathcal{NF}_2}(m)}.$$

In other words, we measure the ratio of the least amount of information in instances of $\mathcal{NF}_1$- and $\mathcal{NF}_2$-schemas, which tells us how much better $\mathcal{NF}_1$ can be compared to $\mathcal{NF}_2$.

Let $\textsc{All}$ be the set of all (unnormalized) schemas.

Theorem 3 (see [12]). *For every $m > 2$:*

- $\textsc{Gain}_{\text{3NF}/\textsc{All}}(m) = 2$;
- $\textsc{Gain}_{\text{BCNF}/\text{3NF}^+}(m) = 2$;
- $\textsc{Gain}_{\text{3NF}^+/\textsc{All}}(m) = 2^{m-2}$.

Stated informally, an arbitrary 3NF design is at least twice as good as not doing any normalization at all. Furthermore, good 3NF designs are much better – in the worst case, they are within a constant factor of two of the best BCNF designs in terms of the amount of redundancy, while guaranteeing dependency-preservation.

5 Applying the measure: XML designs

The information theoretic measure is very robust. It lets us justify relational normal forms and go beyond them – we saw how to use it for normal form comparison, for example. It was also shown in [2] how to reason about normalization algorithms using the information-theoretic measure.

We now switch our attention to XML, and show that the same information-theoretic techniques give us a notion of redundancy-eliminating normal form, which is an analog of BCNF, as well as a notion of "second best" form, similar to 3NF. Namely, we shall do the following.

1. We define functional dependencies for XML, for specifying constraints.
2. We then show that the information-theoretic measure applies to XML documents.
3. We present the notion of a redundancy-eliminating normal form, called XNF, and sketch an algorithm for converting XML designs into XNF.
4. Finally, we analyze good XML designs assuming XML documents are shredded into relations. Again, we show that XNF eliminates redundancies.

We start with the notion of functional dependency. An analog of a relational attribute in the case of XML is a *path* through a DTD, i.e. a sequence of labels consistent with the DTD of a document. For example, if we represent a relational schema with attributes ABC and FDs $AB \to C$ and $C \to A$ by a DTD D:

$$r \to tuple^* \tag{1}$$

with *tuple* having attributes $@A, @B, @C$, respectively, then the FDs will be represented as follows:

$$\begin{array}{rcl} \{r.tuple.@A,\ r.tuple.@B\} & \to & r.tuple.@C \\ r.tuple.@C & \to & r.tuple.@A \end{array} \tag{2}$$

If we look at the DTD D' for storing information about conferences and papers:

$$\begin{array}{c} db \to conf^* \\ conf \to paper^* \end{array} \tag{3}$$

where the element type *paper* comes with attributes $@author, @title, @year$, then we have an FD

$$db.conf \quad \to \quad db.conf.paper.@year \tag{4}$$

For example, it is natural to expect that all the papers appearing in XSym 2007 will have 2007 as the year of their publication.

The notion of satisfaction of FDs should be intuitively clear from these examples; the interested reader is referred to [1] for a formal definition that relies on a notion of tree tuples.

Now we have XML schemas which are given by DTDs D and sets of XML FDs F. If we have a document T that conforms to D and satisfies constraints in

F, we can define a set of *positions* in that document as the set of all the places where an attribute value occurs (more precisely, as sets of pairs $(v, @l)$, where v is a node identifier in a tree, and $@l$ is an attribute associated with that node). Once we have the notion of positions, we can define the measure

$$\text{RIC}_T(p|F)$$

in exactly the same way as we defined $\text{RIC}_R(p|F)$, except that the we use the set of positions in a document T as opposed to the set of positions in relation R. It satisfies all the same basic properties as $\text{RIC}_R(p|F)$.

We then say that an XML specification (D, F) is *well-designed* if for every document T that conforms to D and satisfies F, and every position p in T, we have $\text{RIC}_T(p|F) = 1$.

How do we characterize the notion of being well-designed? To see what a reasonable normal form for XML might be, we analyze the examples shown above. Given the DTD (1) and FDs (2), let us look at the FD $r.tuple.@C \rightarrow r.tuple.@A$. The left-hand side implies $r.tuple.@A$, but since $@C$ is not a key, the left-hand side does *not* imply $r.tuple$ – indeed, $@C$ does not determine the tuple uniquely.

Looking at the DTD (3) and the FD (4), we see that the left-hand side of (4) does not imply *db.conf.paper* (as this would mean that there was a single paper published in the proceedings!).

So what is common to these examples? In both cases we have:

- redundancy built into the specification – in the first case it is essentially a non-BCNF relational design, and in the second case the attribute $@year$ is stored multiple times;
- an FD of the form $X \rightarrow path.@l$ such that $X \rightarrow path$ is *not* implied by other FDs.

This was the motivation for the following definition given in [1].

Definition 2. *An XML specification (D, F) given by a DTD D and a set F of FDs is in the* XML Normal Form (XNF) *iff for every FD $X \rightarrow path.@l$ implied by F, it is the case that $X \rightarrow path$ is also implied by F.*

This turns out to be the definition suggested by the information-theoretic approach.

Theorem 4 (see [2]). *An XML specification (D, F) given by a DTD D and a set F of FDs is well-designed iff it is in XNF.*

Two specifications seen earlier are *not* in XNF. A natural question is then how to convert them into XNF. Looking at the first example, we have to apply the usual relational decomposition; for example, a new DTD will look like

$$r \rightarrow tuple^*, r_{new}$$
$$r_{new} \rightarrow tuple^*_{new}$$

where *tuple* has attributes $@A, @C$, and $tuple_{new}$ has attributes $@B, @C$. We put in the FD $r.tuple.@C \rightarrow r.tuple$, since $@C$ is a key, if *tuple* contains only $@A$ and $@C$ as attributes.

In the second example, we do not make a relational split, but instead simply make $@year$ an attribute of *conf*, which eliminates the violation of XNF seen above.

An algorithm for converting a specification into XNF is essentially this:

> keep applying the "relational split" and the "hierarchical attribute move" steps illustrated above.

One can prove [1] that such an application of two basic transformation rules results in an XNF design.

So far we made an assumption that XML documents are represented as tree-structures. Very often, however, documents are shredded into relations [16]. One of the most common techniques for storing XML in relational databases is *inlining* [16]. The idea is that separate relations are created for element types that appear under a Kleene star, and all other element types are inlined in the relations corresponding to their parents. Each relation for an element type has an id attribute that is a key for that relation, as well as a parent id attribute that is a foreign key pointing for the parent of that element. All the attributes of a given element type in the DTD become attributes in the relation corresponding to that element type.

For example, the relational schema for storing XML documents conforming to the DTD in (3) would be

$$\begin{array}{l} conf(\underline{confID}, name) \\ paper(\underline{paperID}, confID, title, author, year), \end{array}$$

assuming that the *conf* element type has attribute *@name*. Keys are underlined.

It is known [13] that XML functional dependencies F are translated into more general constraints over the inlined relational representation, more precisely, into a set Σ_F of equality-generating dependencies. The inlining mapping also associates a position $\delta(p)$ of the relational representation with each position p in the XML document.

Let (S, Σ_F) be an inlining translation (S, Σ_F) of (D, F), where S is a relational schema, and Σ_F is a set of equality-generating dependencies. We say that (D, F) is *well-designed for relational storage* iff for every XML tree T conforming to D and satisfying F and every position p in T, we have $\text{RIC}_{R_T}(\delta(p)|\Sigma_F) = 1$, where R_T is the relational instance of S into which T is transformed.

The next result shows that XNF captures the notion of being well-designed for relational representation of XML documents as well.

Theorem 5 (see [13]). *An XML specification (D, F) is well-designed for relational storage iff it is in XNF.*

Summing up, the following are equivalent for a specification consisting of a DTD D and a set of XML FDs F:

1. (D, F) is well-designed;
2. (D, F) is well-designed for relational storage;
3. (D, F) is in XNF.

We conclude with a simple condition that guarantees "reasonable" designs from the point of view of the information-theoretic measure. First, one can show that for documents not in XNF, the values of both $\text{RIC}_T(p|F)$ and $\text{RIC}_{R_T}(\delta(p)|\Sigma_F)$ can be arbitrarily low [13].

One type of constraints often used for XML documents is *relative*: such constraints do not hold in the entire document, but only in a part of it restricted to descendants of some element type [4, 10]. In the case of XML FDs, we say that an FD $\{q_1, \ldots, q_n\} \to q$ is *relative* if

- for some $i \in [1, n]$, the path q_i ends with an element type (rather than an attribute);
- for all $j \neq i$, the paths q_j extend q_i (in other words, q_i is a prefix of q_j); and
- for some path p which is a prefix of q_i and ends on an element type τ, there exists an element type τ' and a rule $\tau' \to e$ in the DTD such that τ occurs under the scope of a Kleene star in the regular expression e.

Theorem 6 (see [13]). *Let (D, F) be a specification in which every FD violating the XNF condition is relative. Then for every tree T conforming to D and satisfying F and every position p in T, we have*

$$\text{RIC}_{R_T}(\delta(p)|\Sigma_F) \; > \; \frac{1}{2},$$

where R_T is the relational instance into which T is transformed.

Thus, if we design an XML document that might violate XNF but the only violating FDs are relative, then the redundancy of each position in the relational storage of the XML document would not be worse than $\frac{1}{2}$. In other words, this would match the worst-case redundancy of 3NF.

6 Open problems

The information-theoretic approach has completely clarified the situation with good relational designs, and best possible XML designs for both native and relational storage. However, it is not yet entirely clear how to handle non-perfect designs that do not eliminate all redundancies. We provided an example of a sufficient condition that matches the bounds on the measure given by 3NF. However, it is not known whether one can achieve effective normalization with respect to that condition, nor is it known whether the condition guarantees dependency-preservation.

The notion of dependency-preservation itself is much less understood for XML. It was shown in [11] that one can produce XML designs capturing 3NF relational designs that do not have dependency-preserving BCNF decompositions in a way that accounts for all the constraints. This needs to be explored

further, as it opens a possibility of storing relations in XML in a way that eliminates redundancies and guarantees dependency-preservation, even if there is no such relational representation.

Finally, it would be nice to extend the idea of using the information-theoretic framework for reasoning about and comparing different shredding techniques for XML documents.

Acknowledgments This invited talk presents results on the information-theoretic approach to database design that have been obtained jointly with Marcelo Arenas and Solmaz Kolahi. I am very greatful to Marcelo and Solmaz for collaborating with me, and for their comments on this short survey. I gratefully acknowledge the support of the European Commission Marie Curie Excellence grant MEXC-CT-2005-024502 and EPSRC grant E005039.

References

1. M. Arenas, L. Libkin. A normal form for XML documents. *ACM TODS* 29: 195–232 (2004). Extended abstract in *PODS'02*.
2. M. Arenas, L. Libkin. An information-theoretic approach to normal forms for relational and XML data. *J. ACM* 52(2): 246–283 (2005). Extended abstract in *PODS'03*.
3. P. A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM TODS* 1(4): 277–298 (1976).
4. P. Buneman, S. Davidson, W. Fan, C. Hara, W. C. Tan. Keys for XML. In *WWW 2001*, pages 201–210.
5. E. F. Codd. Further normalization of the data base relational model. IBM Research Report, 1971.
6. E. F. Codd. Recent Investigations in Relational Data Base Systems. *IFIP Congress 1974*, pages 1017–1021.
7. M. Dalkilic and E. Robertson. Information dependencies. In *PODS'00*, pages 245–253.
8. D. W. Embley and W. Y. Mok. Developing XML documents with guaranteed "good" properties. In *ER'01*, pages 426–441.
9. R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM TODS*, 2(3):262–278, 1977.
10. W. Fan, L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM* 49(3): 368–406 (2002).
11. S. Kolahi. Dependency-preserving normalization of relational and XML data. *J. Comput. Syst. Sci.* 73(4): 636–647 (2007).
12. S. Kolahi, L. Libkin. On redundancy vs dependency preservation in normalization: an information-theoretic study of 3NF. *PODS 2006*, pages 114–123.
13. S. Kolahi, L. Libkin. XML design for relational storage. *WWW 2007*, pages 1083–1092.
14. T. T. Lee. An information-theoretic analysis of relational databases - Part I: Data dependencies and information metric. *IEEE Trans. on Software Engineering*, 13(10):1049–1061, 1987.
15. Oracle's General Database Design FAQ. `http://www.orafaq.com/faqdesgn.htm`.

16. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *VLDB*, pages 302–314, 1999.
17. M. Vincent, J. Liu. Multivalued dependencies and a 4NF for XML. In *CAiSE* 2003, pages 14–29
18. M. Vincent, J. Liu, C. Liu. Strong functional dependencies and their application to normal forms in XML. *ACM TODS* 29(3): 445–462 (2004).
19. J. Wang, R.Topor. Removing XML data redundancies using functional and equality-generating dependencies. In *ADC 2005*, pages 65–74.
20. C. Zaniolo. A new normal form for the design of relational database schemata. *ACM TODS* 7 (1982), 489–499.